Код на Hamming

Кристиян Стоименов

24 февруари 2024 г.

ТУЕС,
ПВМКС

Какво е код на Hamming? i

- Метод за внасяне на „защитни" битове, чрез които да маркираме съобщение, което изпращаме по такъв начин, че да можем да научим за възникване на до две грешки и да поправим една.
- Измислени от Richard Hamming през 40-те години в Bell Labs.
- Изпращаме съобщения с дължина $k = 2^r - r - 1$, добавяйки точно r „защитни" бита (т.нар. redundency).
- Това означава, че например, можем да боравим с 15-битови *блокове* с 4 бита redundency (11-битови съобщения); или пък със 7-битови блокове с 3 бита redundency (4-битови съобщения).

Какво е код на Hamming? ii

- Тези кодове са известни обичайно съответно като (15, 11, 3) код и (7, 4, 3) код.
- Разбира се, можем да използваме и по-големи блокове (напр. 256b), но поради свойствата му, способностите за брой поправими грешки остават само за единствен бит. Следователно вероятността да изпуснем възникали (четен брой) грешки е правопропорционална на големината на блока.
- Основната идея на кода на Hamming е заложена в *проверката по четност* (parity bit).

Проверка по четност

Припомняме какво представляваше проверката по четност:

- Уговаряме се да следваме правилото във всеки блок данни задължително броят на вдигнатите битове (1) да бъде четен.
- Така, ако блоковете ни са 8-битови с parity bit в края, то съобщението `0110001` ще има parity bit `1`, докато `1111000` - `0`.
- Това е механизмът, който използва `UART` интерфейса, в конфигурацията `8E` (8-битови пакети, „четна" четност).

Код на Hamming - принципи i

Основната идея, която трябва да усвоите е следната: *ако поставим достатъчно на брой parity проверки на съответните места, получаваме удобен и лесен начин за откриване на единствена грешка*.

Ще се опитаме да разберем (15, 11) кода, но принципът на действие се запазва при блок с произволна дължина 2^k.

Код на Hamming - принципи ii

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Използваме този наглед нагласен „формат”, попълвайки съобщението, което искаме да изпратим (11b) в зелените клетки. В жълтите ще поставим redundency битовете, а оражевото временно ще игнорираме.

Код на Hamming - принципи iii

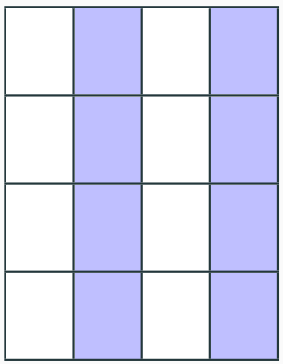
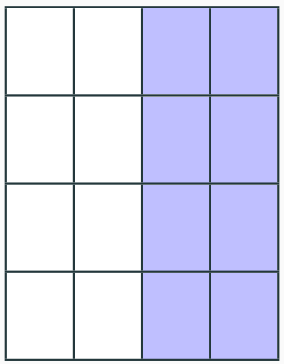
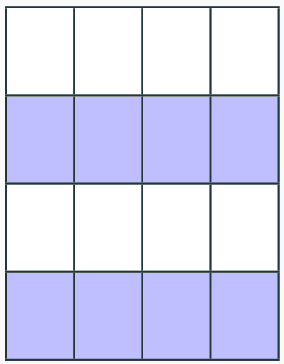
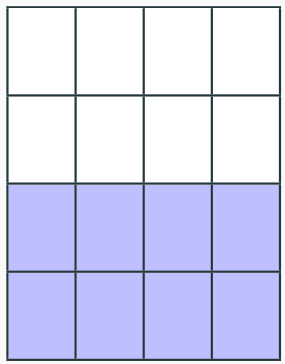
Стойностите, поставени вътре в шаблона, отговарят на „индекса”, който ще използваме за съответната клетка (т.е. аналогично можем да си представим и последователно разположение, вместо таблично).

Код на Hamming - принципи iv

Нека попълним клетките със съобщението 10011001101:

			1
	0	0	1
	1	0	0
1	1	0	1

Жълтите клетки (redundency битовете) попълваме с parity стойности за следните четири подгрупи от клетки:

Код на Hamming - принципи v

Забележете, че първите две подгрупи успяват да намерят колоната, в която е възникнала грешка, докато вторите две - редичката.

Нека определим стойностите на redundency битовете за примерното съобщение:

			1
	0	0	1
	1	0	0
1	1	0	1

			1
	0	0	1
	1	0	0
1	1	0	1

			1
	0	0	1
	1	0	0
1	1	0	1

			1
	0	0	1
	1	0	0
1	1	0	1

Код на Hamming - принципи vi

Както може би забелязвате, интересното за позициите на redundency битовете (и причината за именно този избор), е това, че всеки redundency бит попада в точно една от подгрупи.

Код на Hamming - принципи vii

Друго важно наблюдение ни показва по какъв начин избрахме точно това да са подгрупите - ако разгледаме отново шаблона за попълване с въведени индексите на всяка от клетките ще забележим, че например първата подгрупа включва тези и само тези клетки, чиито индески завършват на 1.

0000	0001	0010	0011
0100	0101	0110	0111
1000	1001	1010	1011
1100	1011	1110	1111

Аналогично следващите подгрупи ни дават тези, включващи `..1.`, `.1..` и `1...`.

Код на Hamming - принципи viii

Така до момента можем да попълним всички без един от битовете в онзи 16-битов пакет:

	1	1	1
0	0	0	1
0	1	0	0
1	1	0	1

Нека го игнорираме още малко и да разгледаме как точно поправяме възникнала грешка.

Бележка: До момента можем да кажем, че сме се запознали с (почти цялата) процедура по кодиране и следва да декодираме.

Код на Hamming - принципи ix

Допускаме, че бит с индекс 6 е бил обърнат.

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

Извършваме проверките по четност, използвайки подгрупите, определящи стойностите на redundency битовете една по една и записваме резултата от всяка от тях като 0 или 1:

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

0

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

1

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

1

	1	1	1
0	0	1	1
0	1	0	0
1	1	0	1

0

Код на Hamming - принципи x

Магия! Получихме именно 6.

Основната причина този декодиращ алгоритъм да работи (и при това така елегетно) се дължи на организацията на „подгрупите" от битове, чрез които ефективно извършваме двоично търсене - най-десният бит грешен ли е? а този вляво? а този вляво? . . . а най-левият бит грешен ли е?

Код на Hamming - принципи xi

Последното парче от пъзела е бит с индекс 0 - каква функция изпълнява той? Всъщност той не е полезен за алгоритъма, който разгледахме до момента. Тъй като обаче обичайно 15b блок данни е твърде необичайно (не е степен на 2), последният бит се използва за проверка по четност на останалите 15 бита. По този начин получаваме механизъм за поправяне на единствена грешка, но улавяне на две възникнали. Тази добавка на още един бит за четност обичайно се нарича разширен код на Hamming.

Код на Hamming - принципи xii

Забележка: Вземете под внимание, че ако възникнат например 4 грешки в единствен блок от данни, кодът на Hamming няма да успее да помогне. Него използваме с допускането, че грешки има не твърде често и не твърде много.

Код на Hamming - обобщение i

Кодираме съобщение като за всеки един от битовете, съставляващи индексите, последователно изчислим бит за четност, а накрая добавим и такъв за целия блок.

Декодираме съобщение като последователно проверим всяка от подгрупите за грешка в четността и си отбележим резултатите от всяка проверка. Накрая събираме резултата и ако той не е 0, поправяме грешката в съответния индекс. Същевременно, проверките извършваме едва когато сме проверили „голямия" parity бит.

Задача

Реализирайте следните функции, които да употребяват разширения (8, 4) Hamming код:

```
uint8_t hamming_encode(uint8_t);
bool hamming_decode(uint8_t *data);
```

Кодирането трябва да приема 4-битово съобщение и да връща 8-битов блок, който да е поставил правилните стойности за redundency битовете.

Декодирането трябва да приема блок данни, кодирани посредством другата функция, и да запази изпратеното съобщение в долните 4 бита от `*data`, поправяйки грешка, ако такава е възникнала.

Върнатата стойност трябва да съответства на резултата от проверката по четност спрямо разширения код на Hamming.

Задача

Използвайки функциите `hamming_encode()` и `hamming_decode()`, изпратете съобщение по I2C, което е защитено посредством разширения (8, 4) Hamming код.

Общо за шумозащитните кодове (*ECC*)

Доста полезни, интересни и *трудни*. Обединяват много дялове на точните науки.

Някои други популярни кодове, които има по-големи поправящи способности включват:

- **Кодове на Reed-Muller** - имат много интересна интерпретация в контекста на крайни геометрии; използвани доста в космоса.
- **Кодове на Reed-Solomon** - CD, DVD, QR, ...
- **BCH кодове** - използват се в памети;
- **Turbo кодове** - 3G, 4G, ...

Литература

- *Hamming codes, part 1*. URL: https://youtu.be/X8jsijhllIA?si=AWhzZjTQMXwF6S4I (дата на посещ. 25.02.2024)
- *Hamming codes, part 2*. URL: https://youtu.be/b3NxrZOu_CE?si=wYRG7U7MYDtfyFUv (дата на посещ. 25.02.2024)
- *Hamming codes*. URL: https://www.youtube.com/watch?v=h0jloehRKas (дата на посещ. 25.02.2024)